An Extended SQL for Temporal Data Management in Clinical Decision-Support Systems

Amar K. Das, Samson W. Tu, Gretchen P. Purcell, Mark A. Musen

Section on Medical Informatics
Stanford University School of Medicine
Stanford, California 94305-5479

ABSTRACT

We are developing a database implementation to support temporal data management for the T-HELPER physician workstation, an advice system for protocol-based care of patients who have HIV disease. To understand the requirements for the temporal database, we have analyzed the types of temporal predicates found in clinical-trial protocols. We extend the standard relational data model in three ways to support these querying requirements. First, we incorporate timestamps into the two-dimensional relational table to store the temporal dimension of both instant- and interval-based data. Second, we develop a set of operations on timepoints and intervals to manipulate timestamped data. Third, we modify the relational query language SQL so that its underlying algebra supports the specified operations on timestamps in relational tables. We show that our temporal extension to SQL meets the temporal data-management needs of protocol-directed decision support.

INTRODUCTION

As database models have incorporated more semantics to interact with clinical decision-support capabilities, the temporal dimension of clinical data has become increasingly important. The need to support timestamped data was recognized early in the development of medical databases, and one of the first implementations of a temporal database was the Time-Oriented Database (TOD), which was modeled after a patient flowsheet [1]. TOD's data model maintains a "cubic view" of medical data by adding a visit-date timestamp as a third dimension to the two dimensions of patient identification and clinical parameters.

Although the cubic view is a common data model underlying many current clinical databases, its representation of timestamps does not meet general clinical data-management requirements for two reasons. First, instant-based data, such as laboratory-test results, have varying granularities. Some results, for example, are time stamped with both date and hour, whereas others have only a date. The cubic view becomes highly irregular when timestamps have granularities other than 1 day. Second, some types of data, such as medication dosages, have temporal durations. The timestamps of these data are best represented as an interval with an explicit start and stop time. The cubic view, however, can not easily represent intervals.

In the T-HELPER (THERAPY HELPER) system for protocol-directed decision support of patients who have HIV disease [2, 3], we plan to overcome the limitations of the cubic view by creating a new temporal data model. In this paper, we present our approach to managing the temporal semantics of medical data in clinical decision-support systems. First, we discuss the types of temporal predicates needed in protocol-directed decision support. Then, we present our approach to representing temporal data in relational databases and to providing temporal operations on these data. Finally, we provide the basis for understanding and executing an extended SQL to query temporal data.

TEMPORAL QUERYING REQUIREMENTS

To determine the requirements for a temporal data model for protocol-directed decision support, we have analyzed the types of *temporal predicates* found in clinical-trial protocols for HIV disease. These predicates require the instantiations of the truth value of a temporal condition in a protocol statement. The predicates can be grouped into different classes:

Temporal Duration. Protocols often require verification of the duration of interval-based data. For example, Triton Bioscience's Study TB01-310188 states that patients should be withdrawn from the study "if treatment with study drugs (rifabutin or placebo) is interrupted *for more than 14 consecutive days*" (Section 7.3, italics added).

Ordinal Ordering. Selection of data based on ordinal ordering is commonly required in protocol-based care. For example, the dose modification section of CCTG Study 522 requires that "if *a second episode* of thrombocytopenia occurs, dosing should be permanently discontinued" (Section 12.133, italics added).

Temporal Context. Context-sensitive queries based on the patient's temporal progression in the study are required in protocol-directed decision support. For example, Stanford University's protocol 20/ID3 stipulates that "phenobarbital may not be given *for the first 3 months of the study*" (Section 9.0, italics added).

In developing a query language to support these predicates in the T-HELPER system, we have chosen to create a temporal version of the Structured Query Language (SQL) for relational databases [4], instead of using an internally developed database language. Commercial relational databases and standard query

0195-4210/92/$5.00

languages (such as SQL) allow researchers to reduce the time of software development, to create portable applications for different hardware platforms, and to support modern client-server architectures [4].

Our approach to extending the relational model to support temporal queries builds upon the research undertaken by others in the database community [5-7]. We first modify the two-dimensional relational table to incorporate the temporal dimension of data; then, we specify the types of temporal operations that are needed for queries; finally, we define a temporal algebra that supports these operations for the modified relations. The algebra is important for understanding an extended SQL, as the database management system translates a specific SQL query into algebraic operations before it optimizes and executes that query.

TEMPORAL DATA MODEL

An algebra is defined as "a set of objects and a collection of operations over those objects" [5]. In the relational model, the objects are tables called *relations*, and the operations take as arguments one or two relations as operands to produce a resultant relation. Since an algebra should be closed (such that operations return the same type of object), a temporal relational database should consist of only one type of relation. In this section, we discuss how we model time, and how we represent timestamped data within a single type of relation.

Representation of Temporal Data

Since time is a continuous value in the real world, it can be represented by real numbers. For database applications, however, time is usually discretized as timepoints on a *timeline*. In our model of time, the timepoints will have only a single granularity, which is at the smallest level of interest in the database application. In outpatient databases, the smallest granularity is typically 1 minute. We can use real-world date–time values to represent timepoints; three special timepoint values are also included in our temporal data model: $+\infty$ (all future timepoints), $-\infty$ (all past timepoints), and now (current timepoint).

We associate three different types of medical data with time: instance-based, interval-based, and time-invariant. We refer to instance-based data as *events*. An event corresponds to a medical datum that occurs instantaneously; a laboratory-test result is an example event. When the timestamp associated with an event is collected, the user may be uncertain of the precise instance at which the event occurred. Instead, she may know only an approximate interval of time during which the event took place. Data models that attempt to capture the temporal uncertainty associated with events normally use a single timepoint with varying granularities (such as minute, hour, and day). The coarser the granularity of the timepoint, the greater the uncertainty regarding the time associated with the event. In our temporal data model, however, all timepoints have the same granularity. We represent an event with a *pair* of timepoints for the lower and upper bounds of the closed *interval of uncertainty* (IOU) during which the user has specified that the event occurred. The lower and upper bounds of the IOU are equal if the user is certain of the timepoint associated with an event. We assume that the probability distribution of the event is uniform over the IOU. We can thus compute the probability of the event over any of the IOU's subinterval as the duration of the subinterval divided by the duration of the IOU.

This representation of events has two advantages over models that use a single timepoint with varying granularities to capture temporal uncertainty [7]. First, our model is not limited to measuring IOUs based on real-world date–time granularities. We can, for example, store a clinical event in a history as occurring sometime during a 30-minute period, instead of using a granularity of 1 hour. Second, when comparing the order of events with overlapping IOUs, we do not need to make timepoint approximations, which data models with different granularities do need to make.

Interval-based data represent *states* and are bounded by start and stop events. An example state is the administration of a clinical-trial protocol, which is initiated by an enrollment event and terminated by a discontinuation event. Both the start and stop events of a state can be represented as IOUs; the closed interval between the upper bound of the start event and the lower bound of the stop event, which we call the *body* of the state, represents an *interval of certainty* (IOC). At each timepoint in the IOC, the value of the state holds true. Time-invariant data are stored as IOCs that exist for all possible timepoints; the values of the timepoints for the start bound are $-\infty$, and those for the stop bound are $+\infty$.

By modeling instant-based, interval-based, and time-invariant data as IOUs, IOCs, or a combination of both, we can define an interval-stamping method for storing these three types within a single type of relational table. We call such a relation a *history*. Each row, or tuple, in a history will store the temporal dimension of a patient parameter over a closed interval; a pair of columns will be required in each history to represent the endpoints of the interval as *timestamp attributes*.

With this method of interval stamping, we can store the IOU of an event within a single tuple, whereas we need three tuples to store the IOUs of a state's start and stop events and the IOC of the body. To distinguish these types of interval, we also use two attributes to maintain the type of interval (*event*, *start event*, *body*, or *stop event*) being stored, and to keep an event's cumulative probability over an IOU (the probability is equal to null if the interval is of type *body*). In storing changes of a patient parameter, we constrain the history so that, at any particular timepoint in the interval, each nontimestamp attribute has a single or null value. We can place this constraint on histories by making the set of timestamp attributes part of the primary key. Figure 1 provides example histories of events and of states.

START_TIME	STOP_TIME	TYPE	PROBABILITY	PATIENT_ID	TEST_NAME	VALUE
Jan 7 1990 12:00PM	Jan 7 1990 12:59PM	event	1.0	8703	WBC	2.20
Jan 7 1990 3:00PM	Jan 7 1990 3:59PM	event	1.0	8703	WBC	2.00
Jan 18 1990 12:00AM	Jan 18 1990 11:59PM	event	1.0	8703	WBC	4.30

(a)

START_TIME	STOP_TIME	TYPE	PROBABILITY	PATIENT_ID	DRUG_NAME	DOSE
Jan 18 1990 12:00AM	Jan 18 1990 11:59PM	start event	1.0	8703	ZIDOVUDINE	400.00
Jan 19 1990 12:00AM	Jan 28 1990 11:59PM	body	null	8703	ZIDOVUDINE	400.00
Jan 29 1990 12:00AM	Jan 29 1990 11:59PM	stop event	1.0	8703	ZIDOVUDINE	400.00
Jan 30 1990 12:00AM	Jan 30 1990 11:59PM	start event	1.0	8703	ZIDOVUDINE	500.00
Jan 31 1990 12:00AM	Jun 17 1990 11:59PM	body	null	8703	ZIDOVUDINE	500.00
Jun 18 1990 12:00AM	Jun 18 1990 11:59PM	stop event	1.0	8703	ZIDOVUDINE	500.00

(b)

Figure 1. The same type of extended relational table, or history, is used to store both events and states. A pair of timestamp attributes (left of the vertical double bar) must be included in each history. The two timestamp attributes are included as part of the primary key (shaded terms). (a) LAB_TEST table: example history of events. (b) MEDICATION table: example history of states.

Operations on Temporal Data

Operations involving events and states are based on comparisons and operations for timepoints and intervals.

Timepoint Comparisons. Few mathematical operations apply directly to discrete timepoints, since we cannot add, multiply, or divide two timepoints. Since all timepoints in our data model have the same granularity, we can compare discrete timepoints to determine a linear order using standard comparison operations. The terms AT, BEFORE, and AFTER are used to indicate the comparison operators =, <, and >, respectively. We can also use the comparison *t* ADJACENT *u* , where *t* and *u* are timepoints, to determine if *t* is 1 granule less than *u*.

Interval Comparisons. Since intervals are represented as pairs of timepoints, comparisons between intervals are based on timepoint comparisons of the upper and lower bounds. A complex set of comparisons — BEFORE, UNTIL, LEADS, STARTS, EQUALS, DURING, SPANS, FINISHES, LAGS, FROM, and AFTER — can be made between two intervals. This set was originally defined by Allen [8].

Timepoint Operators. In addition to comparing timepoints, we can subtract one timepoint from another to find the length of time between them. The function DURATION(t_1, t_2) returns the number of granules between the timepoints t_1 and t_2. We can also create new timepoints by adding (or subtracting) granules to a given timepoint. We use ADDGRANULE(*t*, *x*) to add to *t* *x* granules to create a new timepoint.

Interval Operators. Given two overlapping intervals, we can compute a new interval as the union, difference, or intersection of the two [9]. The operation CATENATE takes two overlapping (or adjacent) intervals and returns a new interval that consists of the set of timepoints in either of the original intervals. The function SHORTEN takes two intervals that overlap and removes from the first interval the points common to both to create a new interval. EXTRACT also takes two intervals, but returns a new interval containing only those points that are in both intervals. The intervals used in these three binary functions can be both IOUs, both IOCs, or a pair of each.

TEMPORAL QUERY LANGUAGE

In this section, we show how operations on temporal data are implemented in an extended SQL.

Temporal Query Syntax

Since we have extended the relational data model to maintain the underlying view of temporal data as timepoints on a timeline, we call the extended SQL **TIMELINESQL (TLSQL).** TLSQL is based on the simple structural framework of SQL, with syntactic extensions to support operations on events and states in histories. TLSQL adds the following new constructs to standard SQL:

1. Temporal abstractions — such as timepoint, interval, and duration — as selectable attributes
2. Selections based on ordinal ordering of timepoints using the terms `FIRST`, `SECOND`, `THIRD`, or `LAST` in the `SELECT` clause
3. Selections based on temporal comparisons of timepoints and intervals using terms in a `WHEN` clause
4. Methods to perform catenation of overlapping or adjoining intervals with the same nontimestamp values using the term `COALESCED` in the `SELECT` clause

5. Methods to perform extraction or subtraction of overlapping intervals in two different histories using the terms CONCURRENT WITH (for EXTRACT) or NOT CURRENT WITH (for SUBTRACT) in the WHEN clause

The formal syntax of a TLSQL retrieval statement contains the following clauses:

```
SELECT [FIRST|SECOND|THIRD|LAST] [COALESCED]
       select_item_commalist
FROM   table_name_commalist
WHEN   temporal_comparison_list
WHERE  search_condition_list
```

Temporal Query Semantics

Although the syntax of TLSQL is an extension of standard SQL syntax, the semantics of a TLSQL query are based solely on the temporal relational model outlined in this paper. The temporal semantics contained in a TLSQL query, consequently, cannot be translated into standard relational algebra [10]. We instead modify and extend standard relational algebra to create a version that incorporates temporal operations on timepoints and intervals. We refer to this version as *temporal relational algebra*. We now describe how temporal projection, selection, and join are a set of algebraic operators that support the temporal querying requirements discussed previously.

Temporal Projection. Standard projection restricts attributes in a table to those specified in the SELECT clause. Temporal projection is similar to standard projection, except that the restriction applies to only the *nontimestamp* attributes. Both timestamp attributes cannot be excluded in the resultant history, because these attributes maintain the temporal dimension of the data. In the following example TLSQL query, we would like to determine how long patient 8073 in our sample database (in Figure 1b) was taking the medication zidovudine, regardless of the uncertainty associated with start and stop events or the particular dosage administered:

```
SELECT COALESCED DRUG_NAME
FROM   MEDICATION
WHERE  PATIENT_ID = 8073 AND
       DRUG_NAME = "ZIDOVUDINE"
```

The query returns the following history with one tuple:

START_TIME	STOP_TIME	DRUG_NAME
Jan 18 1990 12:00AM	Jun 18 1990 11:59PM	ZIDOVUDINE

By using temporal projection and the term COALESCED, we can specify which attributes should be selected and which adjoining intervals should be catenated into a single interval in the resultant relation. This algebraic operation provides the basis for reasoning on the temporal duration of specified attributes.

Temporal Selection. Standard selection restricts which tuples from the operand relation will be included in the resultant relation; the tuples in the result must satisfy the predicates in the WHERE clause. In our temporal data model, we are also interested in restricting tuples based on temporal comparisons and ordinal selection. We add the terms FIRST, SECOND, THIRD, and LAST to specify ordinal selection, and a WHEN clause to specify the temporal comparisons that should be used. The following example TLSQL query requests the second white-blood-cell count on January 7, 1990 for patient 8073:

```
SELECT SECOND VALUE
FROM   LAB_TEST
WHEN   (START_TIME, STOP_TIME) DURING
       (Jan 7 1990 12:00AM, Jan 7 1990 11:59PM)
WHERE  PATIENT_ID = 8073 AND TEST_NAME = "WBC"
```

The query returns the following history with one tuple:

START_TIME	STOP_TIME	VALUE
Jan 7 1990 3:00PM	Jan 7 1990 3:59PM	2.00

Temporal selection of data provides an algebraic operation to determine whether data match the ordinal-selection and temporal-comparison predicates in temporal-reasoning criteria.

Temporal Join. The ability to combine data from two different relations is an important feature of relational algebra that is provided in the join operator. We define a temporal version of join that can combine timestamped data from two different histories into a single history. The intervals in the resultant history are created by applying the EXTRACT operation to overlapping intervals of the operand histories; the nontimestamp attributes are taken from both operand relations. In the following TLSQL query, for example, we would like to determine the values of the white-blood-cell counts when patient 8073 was taking the medication zidovudine:

```
SELECT LAB_TEST.VALUE
FROM   LAB_TEST, MEDICATION
WHEN   LAB_TEST CONCURRENT WITH MEDICATION
WHERE  LAB_TEST.PATIENT_ID = 8073 AND
       LAB_TEST.PATIENT_ID =
       MEDICATION.PATIENT_ID AND
       LAB_TEST.TEST_NAME = "WBC" AND
       MEDICATION.DRUG_NAME = "ZIDOVUDINE"
```

The query returns the following history with one tuple:

START_TIME	STOP_TIME	VALUE
Jan 18 1990 12:00AM	Jan 18 1990 11:59PM	4.30

Using the temporal join operator, we can create new histories that provide temporal contexts during which other temporal predicates can be specified.

Besides temporal projection, selection, and join, we also provide other temporal versions of standard relational algebra; the complete set of temporal relational algebra is provided elsewhere [11]. The relatively simple TLSQL

query examples in this section, however, demonstrate the expressive power of the language and the ability to support temporal predicates.

DISCUSSION

In this paper, we have provided our approach to modeling the temporal semantics of medical data with the relational data model. Our temporal model of relational databases consists of relational tables called *histories*, which store either instant-based *events* or interval-based *states*. We also define a set of temporal operations for the timepoints and intervals used in event and state representation.

The temporal data model uses two timestamp attributes to represent the temporal changes of each patient parameter; it consequently uses space less efficiently than do data models that use only one timestamp attribute (such as TOD). The semantics of intervals and of granularities associated with medical data, however, are not captured by the latter models. As the price of hard-disk storage continues to decrease, the cost in space requirements for our temporal data model will be outweighed by the benefits in representation flexibility that the model provides.

We have also discussed a method to create a temporal version of the relational query language, SQL. The difficulty of making temporal queries in standard SQL has been encountered by other researchers in medical informatics [12]. We are able to overcome these difficulties by developing a temporal relational algebra for histories based on our set of temporal operations. We extend the syntax of SQL with new terms, so that the semantics of an SQL for histories (TLSQL) can be understood in terms of the underlying algebraic operators. The example TLSQL queries illustrate that the set of algebraic operators supports the three classes of temporal predicates we found in clinical-trial protocols.

We are not certain, however, of the range of temporal predicates that might be required by clinical decision-support systems. Our current research in the theory of temporal databases is, consequently, focused on defining an algebra that has *temporal completeness*. The database research community has yet to reach a consensus on the issue of temporal completeness [5]; however, by reviewing the types of temporal predicates found in clinical-trial protocols and in other temporal query languages, we can at least gain an understanding of what is required for temporal completeness in the medical applications that we most want to automate.

We are currently implementing our temporal data model and TLSQL with a Sybase relational database and C host language. The implementation, called Chronus, provides both preprocessing of TLSQL into standard SQL operators and postprocessing of data for operations not supported directly by standard relational algebra. By supporting and maintaining an event- and state-oriented view of patient data in the simple structure of relational tables, Chronus permits clinical decision support systems to take advantage of both the semantics of timestamped data and the properties of relational databases.

Acknowledgments

We thank L. Dupré for her editorial assistance in the preparation of this paper. This work has been supported in part by grant HS06330 from the Agency for Health Care Policy and Research and by grants LM05208 and LM07033 from the National Library of Medicine. Dr. Musen is a recipient of an NSF Young Investigator Award.

References

[1] Wiederhold, G., Fries, J.F., and Weyl, S. Structured organization of clinical data bases. *Proceedings of AFIPS NCC*. AFIPS, 1975:479–85.

[2] Musen, M.A., Carlson, R.W., Fagan, L.W., Deresinski, S.C., and Shortliffe, E.H. Computer support for community-based clinical research. *Proceedings of the Sixteenth Annual SCAMC*. Washington, DC. November 1992.

[3] Shahar, Y., Tu, S.W., Das, A.K., and Musen, M.A. A problem-solving architecture for managing temporal data and their abstractions. *Proceedings of the Workshop on Implementing Temporal Reasoning, AAAI-92*, San Jose, CA. July 1992.

[4] Date, C.J. *A Guide to the SQL Standard*. Reading, MA: Addison-Wesley, 1989.

[5] McKenzie, L.E., and Snodgrass, R.T. Evaluation of relational algebra incorporating the time dimension in databases. *ACM Computing Survey* 23:501–43, 1991.

[6] Navathe, S.B., and Ahmed, R. A temporal relational model and a query language. *Information Sciences* 49:147–75, 1989.

[7] Sarda, N.L. Extensions to SQL for historical databases. *IEEE TKDE* 2:220–30, 1990.

[8] Allen, J.F. Maintaining knowledge about temporal intervals. *Comm ACM* 26:832–43, 1983.

[9] Wiederhold, G. *Semantic Database Design*. New York: McGraw-Hill (in press).

[10] Ullman, J.D. *Principles of Databases*. Rockville, MD: Computer Science Press, 1984.

[11] Das, A.K., Tu. S.W., and Musen, M.A. A historical relational data model for managing temporal data. Tech. Rep. KSL-91-77, Knowledge Systems Laboratory, Stanford University, Stanford, CA. 1992.

[12] Huff, S.H., Berthelson, C.L., and Pryor, T.A. Evaluation of an SQL model of the HELP patient database. *Proceedings of the Fifteenth Annual SCAMC*. Washington, DC. P.D. Clayton (Ed.) IEEE Computer Society. November 1991.